It's Like Python But: Towards Supporting Transfer of Programming Language Knowledge

Nischal Shrestha
NC State University
Raleigh, NC, USA
nshrest@ncsu.edu

Titus Barik
Microsoft
Redmond, WA, USA
titus.barik@microsoft.com

Chris Parnin
NC State University
Raleigh, NC, USA
cjparnin@ncsu.edu

***Abstract*—Expertise in programming traditionally assumes a binary novice-expert divide. Learning resources typically target programmers who are learning programming for the first time, or expert programmers for that language. An underrepresented, yet important group of programmers are those that are experienced in one programming language, but desire to author code in a different language. For this scenario, we postulate that an effective form of feedback is presented as a transfer from concepts in the first language to the second. Current programming environments do not support this form of feedback.**

In this study, we apply the theory of learning transfer to teach a language that programmers are less familiar with—such as R—in terms of a programming language they already know––such as Python. We investigate learning transfer using a new tool called Transfer Tutor that presents explanations for R code in terms of the equivalent Python code. Our study found that participants leveraged learning transfer as a cognitive strategy, even when unprompted. Participants found Transfer Tutor to be useful across a number of affordances like stepping through and highlighting facts that may have been missed or misunderstood. However, participants were reluctant to accept facts without code execution or sometimes had difficulty reading explanations that are verbose or complex. These results provide guidance for future designs and research directions that can support learning transfer when learning new programming languages.

I. Introduction

Programmers are expected to be fluent in multiple programming languages. When a programmer switches to a new project or job, there is a ramp-up problem where they need to become proficient in a new language [1]. For example, if a programmer was proficient in Python, but needed to learn R, they would need to consult numerous learning resources such as documentation, code examples, and training lessons. Unfortunately, current learning resources typically do not take advantage of a programmer's existing knowledge and instead present material as if they were a novice programmer [2]. This style of presentation does not support experienced programmers [3] who are already proficient in one or more languages and harms their ability to learn effectively and efficiently [4].

Furthermore, the new language may contain many inconsistencies and differences to previous languages which actively inhibit learning. For example, several blogs and books [5] have been written for those who have become frustrated or confused with the R programming language. In an online document [6], Smith lists numerous differences of R from other high-level languages which can confuse programmers such as the following:

> *Sequence indexing is base-one. Accessing the zeroth element does not give an error but is never useful.*

In this paper, we explore supporting learning of programming languages through the lens of *learning transfer*, which occurs when learning in one context either enhances (positive transfer) or undermines (negative transfer) a related performance in another context [7]. Past research has explored transfer of cognitive skills across programming tasks like comprehension, coding and debugging [8], [9], [10]. There has also been research exploring the various difficulties of learning new programming languages [11], [12] and identifying programming misconceptions held by novices [13]. However, limited research has focused on the difficulties of learning languages for experienced programmers and the interactions and tools necessary to support transfer.

To learn how to support transfer, we built a new training tool called Transfer Tutor that guides programmers through code snippets of two programming languages and highlights reusable concepts from a familiar language to learn a new language. Transfer Tutor also warns programmers about potential misconceptions carried over from the previous language [14].

We conducted a user study of Transfer Tutor with 20 participants from a graduate Computer Science course at North Carolina State University. A qualitative analysis on think-aloud protocols revealed that participants made use of learning transfer even without explicit guidance. According to the responses to a user satisfaction survey, participants found several features useful when learning R, such as making analogies to Python syntax and semantics. However, participants also pointed out that Transfer Tutor lacks code executability and brevity. Despite these limitations, we believe a learning transfer tool can be successful in supporting expert learning of programming languages, as well as other idioms within the same language. We discuss future applications of learning transfer in other software engineering contexts, such as assisting in code translation tasks and generating documentation for programming languages.

II. Motivating Example

Consider Trevor, a Python programmer who needs to switch to R for his new job as a data analyst. Trevor takes an online

978-1-5386-4235-1/18/$31.00 ©2018 IEEE

```python
df = pd.read_csv('Questions.csv')
df = df[df.Score > 0][0:5]
```

(a) Python

```r
df <- read.csv('Questions.csv')
df <- df[df$Score > 0, ][1:5, ]
```

(b) R

Fig. 1. (a) Python code for reading data, filtering for positive scores and selecting 5 rows. (b) The equivalent code in R.

course on R, but quickly becomes frustrated as the course presents material as if he is a novice programmer and does not make use of his programming experience with Python and Pandas, a data analysis library. Now, Trevor finds himself ill-equipped to get started on his first task at his job, tidying data on popular questions retrieved from Stack Overflow (SO), a question-and-answer (Q&A) community [15]. Even though he is able to map some concepts over from Python, he experiences difficulty understanding the new syntax due to his Python habits and the inconsistencies of R. Trevor asks help from Julie, a seasoned R programmer, by asking her to review his R script (see Fig. 1) so he can learn proper R syntax and semantics.

Trevor's task is to conduct a typical data analysis activity, tidying data. He is tasked with the following: 1) read in a comma-separated value (csv) file containing Stack Overflow questions 2) filter the data according to positive scores and 3) select the top five rows. Julie walks him through his Python code and explains how they relate to the equivalent code she wrote in R.

Julie teaches Trevor that R has several assignment operators that he can use to assign values to variables but tells him that the `<-` syntax is commonly used by the R community. However, she tells him that the `=` operator can also be used in R just like Python. To read a csv file, Julie instructs Trevor to use a built-in function called `read.csv()` which is quite similar to Python's `read_csv()` function.

Moving on to the next line, Julie explains that selecting rows and columns in R is very similar to Python with some subtle differences. The first subtle difference that she points out is that when subsetting (selecting) rows or columns from a data frame in Python, using the `[` syntax selects rows. However, using the same operator in R will select columns. Julie explains that the equivalent effect of selecting rows works if a comma is inserted after the row selection and the right side of the comma is left empty (Figure 1b). Julie tells him that since the right side is for selecting columns, leaving it empty tells R to select all the columns. To reference a column of a data frame in R, Julie explains that it works almost the same way as in Python, except the `.` (dot) must be replaced with a `$` instead. Finally, Julie points out that R's indexing is 1-based, so the range for selecting the five rows must start with 1, and unlike Python, the end index is inclusive. Trevor now has some basic knowledge of R. Could tools help Trevor in the same way Julie was able to?

III. Transfer Tutor

A. Design Rationale

We created a new training tool called Transfer Tutor that takes the place of an expert like Julie and makes use of learning transfer to teach a new programming language. Transfer Tutor teaches R syntax and semantics in terms of Python to help provide scaffolding [16] so programmers can start learning from a familiar context and reuse their existing knowledge. Our approach is to illustrate similarities and differences between code snippets in Python and R with the use of highlights on syntax elements and different types of explanations.

We designed Transfer Tutor as an interactive tool to promote "learnable programming" [17] so that users can focus on a single syntax element at a time and be able to step through the code snippets on their own pace. We made the following design decisions to teach data frame manipulations in R: 1) highlighting similarities between syntax elements in the two languages 2) explicit tutoring on potential misconceptions and 3) stepping through and highlighting elements incrementally.

B. Learning Transfer

Transfer Tutor supports learning transfer through these feedback mechanisms in the interface:

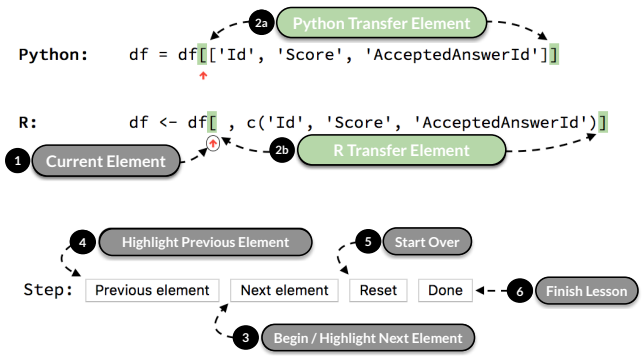
- **Negative Transfer:** 'Gotchas' warn programmers about a syntax or concept that either does not work in the new language or carries a different meaning and therefore should be avoided.
- **Positive Transfer:** 'Transfer' explanations describe a syntax or concept that maps over to the new language.
- **New Fact:** 'New facts' describe a syntax or concept that has little to no mapping to the previous language.

Each type of feedback consists of a highlighted portion of the code in the associated language (Python or R) with its respective explanation, which serves as affordances for transfer [18]. Furthermore, we support deliberate connections between elements, by allowing participants to step through the code, which helps them make a mindful abstraction of the concepts [19]. Finally, we focus on transferring declarative knowledge [20], such as syntax rules, rather than procedural knowledge, such as general problem-solving strategies.

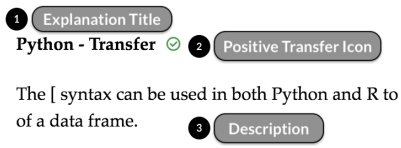
C. User Experience

This section presents screenshots of Transfer Tutor and a use case scenario. The user experience of Transfer Tutor is presented from the perspective of Trevor who decides to use the tool to learn how to select columns of a data frame in R, a 2D rectangular data structure which is also used in Python/Pandas. The arrows and text labels are used to annotate the various features of the tool and are not actually presented to the users.

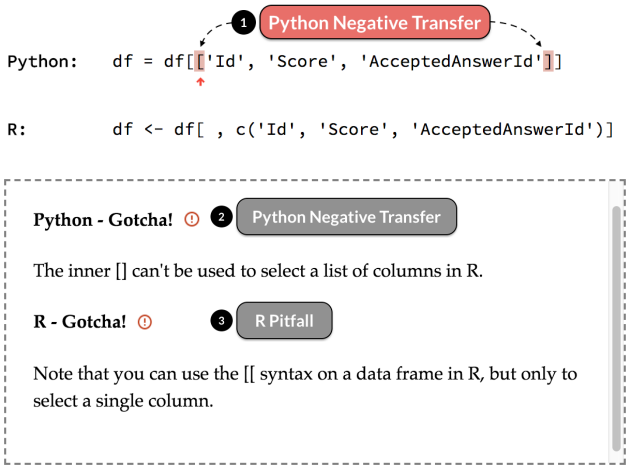
1) Code Snippets and Highlighting: Trevor opens up Transfer Tutor and notices that the tool displays two lines of code, where the top line is Python, the language that he is already familiar with and on the bottom is the language to learn which is R. Trevor examines the stepper buttons below the snippets and clicks ❸ which begins the lesson and highlights some syntax elements:



Travis notices ❶ points to the current syntax element in Python and R indicated by ❷a and ❷b. Trevor looks over to the right at the explanation box:

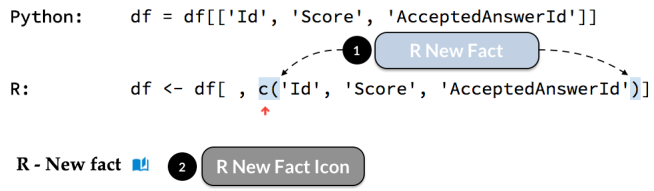


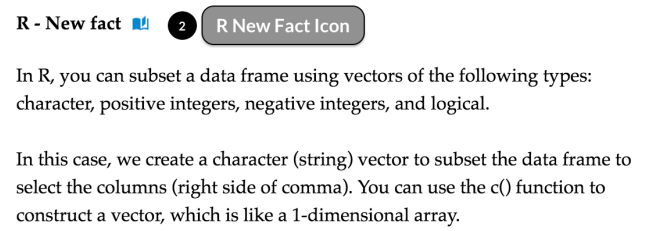
2) Explanation Box: Trevor sees ❶ which refers to a Python 'transfer' with ❷ showing the transfer icon. He reads ❸ and learns that the [operator can be used in R. Transfer Tutor treats this syntax as a positive transfer since it can be reused. Trevor moves on to the next element:



Trevor looks at ❶ which is a red highlight on the Python code. He reads ❷ in the explanation box for clarification.

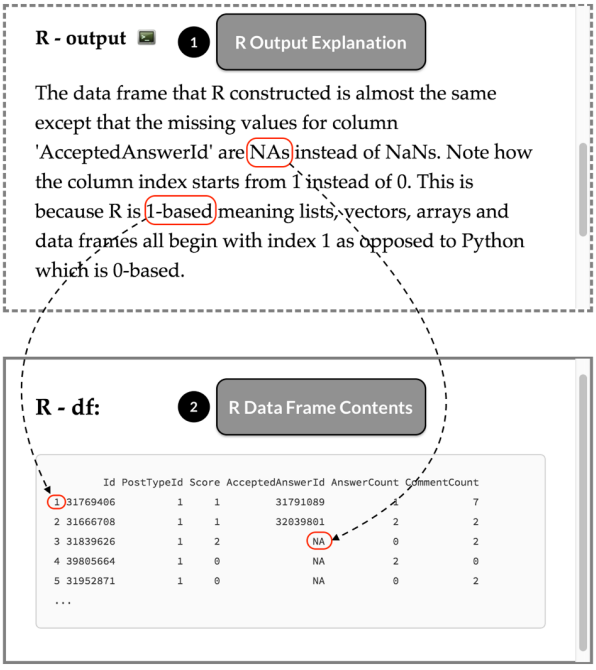
Trevor learns about a Python 'gotcha': the `[[` syntax from Python can't be used in R. Trevor then reads ❸ which explains an R 'gotcha' about how the `[[` syntax is legal in R, but semantically different from the Python syntax as it only selects a single column. In this case, Transfer Tutor warns him about a subtle difference, a negative transfer that could cause him issues in R. Trevor moves on to the next element and examines the elements that are highlighted blue:





Trevor looks at ❶ then ❷ and realizes he's looking at a 'new fact' about R. Transfer Tutor describes the `c()` function used to create a vector in R, which doesn't have a direct mapping to a Python syntax.

3) Code Output Box: Finally, Trevor steps through the code to the end, and the code output box now appear at the bottom which displays the state of the data frame:



Trevor reads ❶ and inspects ❷ to understand the contents of the data frame in R and how it differs from Python's data frame: 1) `NaNs` from Python are represented as `NAs` and 2) Row

indices start from 1 as opposed to 0. Transfer Tutor makes it clear that selecting columns of a data frame in R is similar to Python with some minor but important differences.

IV. Methodology

A. Research Questions

We investigated three research questions using Transfer Tutor to: 1) determine face validity of teaching a new language using an interactive tool 2) examine how programmers use Transfer Tutor and 3) determine which affordances they found to being useful for learning a new language.

RQ1: Are programmers learning R through Transfer Tutor? To identify if training through learning transfer is an effective approach in the context of programming, this question is used to determine the face validity of Transfer Tutor's ability to teach R.

RQ2: How do programmers use Transfer Tutor? Investigating how programmers use Transfer Tutor can identify when it supports learning transfer, and whether the affordances in the tool align with the way programmers reason about the problem.

RQ3: How satisfied are programmers with learning R when using the Transfer Tutor? We want to learn what features of Transfer Tutor programmers felt were useful to them. If programmers are satisfied with the tool and find it useful, it is more likely to be used.

B. Study Protocol

1) Participants: We recruited 20 participants from a graduate Computer Science course at our University, purposely sampling for participants with experience in Python, but not R. We chose to teach R for Python programmers because both languages are used for data science programming tasks, yet have have subtle differences that are known to perplex novice R programmers with a background in Python [5], [6], [21].

Through an initial screening questionnaire, participants reported programming experience and demographics. Participants reported their experience with Python programming with a median of "1-3 years" (7), on a 4-point Likert-type item scale ranging from "Less than 6 months", "1-3 years", "3-5 years", and "5 years or more" (▮▪▫▫). Participants reported a median of "Less than 6 months" (19) of experience with R programming (▮▫▫▫), and reported a medium of "1-3 years" with data analysis activities (▮▪▫▫). 16 participants reported their gender as male, and four as female; the average age of participants was 25 years (sd = 5).

All participants conducted the experiment in a controlled lab environment on campus, within a 1-hour time block. The first author of the paper conducted the study.

2) Onboarding: Participants consented before participating in the study. They were presented with a general instructions screen which described the format of the study and familiarized them with the interface. The participants then completed a pre-test consisting of seven multiple choice or multiple answer questions, to assess prior knowledge on R programming constructs for tasks relating to indexing, slicing, and subsetting of data frames. The questions were drawn from our own expertise in the language and quizzes from an online text.[1] The presentation of questions was randomized to mitigate ordering effects. We also asked participants to think-aloud during the study, and recorded these think-aloud remarks as memos.

3) Study Materials: The authors designed four lessons on the topic of data frame manipulation, where each lesson consists of a one line code snippet in both languages and explanations associated with the relevant syntax elements. The authors also designed questions for the pre-test and post-test (see Table I). Finally, the authors designed a user satisfaction survey of Transfer Tutor. The study materials are available online.[2]

4) Tasks: Participants completed the following lessons on R: 1) assignment and reading data, 2) selecting columns, 3) filtering, and 4) selecting rows and sorting. Participants stepped through each lesson as described in Section III. Within each lesson, participants interacted with 5–8 highlights and corresponding explanation boxes.

5) Wrap-up: At the end of the study, participants completed a post-test containing the same questions as the pre-test. Participants completed a user satisfaction survey asking the participants for additional feedback on the tool. The survey asked them to rate statements about the usefulness of the tool using a 5-point Likert scale. These statements targeted different features of the tool such as whether or not highlighting syntax elements was useful for learning R. The survey also contained free-form questions for feedback regarding the tool such as the most positive and negative aspects, how they could benefit from using the tool and what features they would add to make it more useful. Finally, participants were given the opportunity to debrief for any general questions they may have had about the study.

C. Analysis

RQ1: Are programmers learning R through Transfer Tutor? We used differences in pre-test and post-test performance as a proxy measure for learning. We assigned equal weight to each question, with each question being marked as incorrect (0 points) or correct (1 point), allowing us to treat them as ordinal values. For the multiple answer questions, the participants received credit if they choose all the correct answers. A Wilcoxon signed-rank test between the participants' pre-test and post-test scores was computed to identify if the score differences were significant (α = 0.05).

RQ2: How do programmers use Transfer Tutor? All authors of the paper jointly conducted an open card sort—a qualitative technique for discovering structure from an unsorted list of statements [22]. Our card sorting process consisted of two phases: *preparation* and *execution*. In the *preparation* phase, we extracted the think-aloud and observational data from the written memos into individual cards, with each card containing

[1] http://adv-r.had.co.nz, chapters "Data Structures" and "Subsetting."

[2] https://github.com/alt-code/Research/tree/master/TransferTutor

TABLE I
PRE-TEST AND POST-TEST QUESTIONS

ID	Question Text	Tot.[1]	Δ[2]
1	Select all the valid ways of assigning a 1 to a variable 'x' in R.	0	18
2	Select all the valid vector types that can be used to subset a data frame.	13	2
3	How would one check if 'x' is the value NA?	0	20
4	Given a data frame df with column indices 1, 2, and 3, which one of these will cause an error?	10	3
5	Which one of these correctly selects the first row of a data frame df?	0	20
6	Which one of these correctly subsets the first five rows and the first column of a data frame df and returns the result as a data frame?	0	18
7	All of these statements correctly select the column 'c' from a data frame df *except*	0	1

[1] Total number of participants who answered correctly in pre-test.
[2] Difference in the number of participants who answered correctly in pre-test and post-test.

a statement or participant observation. We labeled each of the cards as either being an indicator of positive transfer, negative transfer, or non-transfer. To do so, we used the following rubric to guide the labeling process:

1) Statements should not be labeled if it includes verbatim or very close reading of the text provided by Transfer Tutor.
2) The statement can be labeled as positive if it demonstrates the participant learning a syntax or concept from Python that can be used in R.
3) The statement can be labeled as negative if it demonstrates the participant learning a syntax or concept in R that is different from Python or breaks their expectation.
4) The statement can be labeled as a non-transfer if it demonstrates the participant encountering a new fact in R for which there is no connection to Python.

In the *execution* phase, we sorted the cards into meaningful themes. The card sort is open because the themes were not pre-defined before the sort. The result of a card sort is not to a ground truth, but rather, one of many possible organizations that help synthesize and explain how programmers interact with tool.

RQ3: How satisfied are programmers with learning R when using Transfer Tutor? We summarized the Likert responses for each of the statements in the user satisfaction survey using basic descriptive statistics. We also report on suggestions provided by participants in the free-form responses for questions, which include suggestions for future tool improvements.

V. RESULTS

In this section we present the results of the study, organized by research question.

A. RQ1: Are programmers learning R after using Transfer Tutor?

All participants had a positive increase in overall score (n = 20). The Wilcoxon signed rank test identified the post-test scores to be significantly higher than the pre-test scores (S = 105, $p < .0001$), and these differences are presented in Table I. Questions 1, 3, 5 and 6 provide strong support for learning transfer. In Question 2 and Question 4, most participants already supplied the correct answer with the pre-test: thus, there was a limited increase in learning transfer. The result of Question 7, however, was unexpected: no participants answered the pre-test question correctly, and there was essentially no learning transfer. We posit potential explanations for this in Limitations (Section VI). Based on these results, using test performance has face validity in demonstrating Transfer Tutor's effectiveness in supporting learning transfer from Python to R.

B. RQ2: How do programmers use Transfer Tutor?

The card sorting results of the observational and think-aloud memos are presented in this section, organized into four findings.

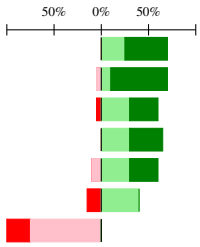
Evidence of using transfer: We collected 398 utterances from our participants during their think-aloud during card sorting. All participants' think-aloud contained utterances related to learning transfer. 35.9% of the total utterances related to transfer, revealing positive (18.9%) and negative transfers (66.4%). They also verbalized or showed behavior to indicate that they were encountering something that was new and didn't map to something they already knew (14%). Other utterances not related to transfer involved verbatim reading of text or reflection on the task or tool.

Participants identified several positive transfers from Python, often without explicit guidance from Transfer Tutor. P4 guessed that the range for selecting a column in the Python code was equivalent to the one in R without Transfer Tutor explicitly mentioning this fact: *"both are the same, 2 colon in Python means 3 in R."* Another participant correctly related Python's dot notation to reference a data frame's column to R's use of dollar sign: *"Oh looks like $ sign is like the dot."* [P17]. This is evidence that programmers are naturally using learning transfer and Transfer Tutor helps support this strategy.

Participants also encountered several negative transfers from either Python or their previous languages. P15 thought the dot in the `read.csv()` function signified a method call and verbalized that the *"read has a csv function"* and later realized the mistake: *"read is not an Object here which I thought it was!"* P5 expressed the same negative transfer, thinking that *"R has a module called read."*. This indicates a negative transfer from object-oriented languages where the dot notation is typically used for a method call.

Participants would also verbalize or show signs of behavior indicating that they have encountered a new fact, or a non-transfer, in R. This behavior occurred before progressing to the element with its associated explanation. P7 encountered the subsetting syntax in R and wondered, *"Why is the left side*

TABLE II
FOLLOW-UP SURVEY RESPONSES

	% Agree	Likert Resp. Counts[1] SD	D	N	A	SA	Distribution[2]
The highlighting feature was useful in learning about R.	95%	0	0	1	5	14	
Stepping through the syntax was useful in learning about R.	79%	0	1	3	2	14	
The explanations that related R back to another language like Python was useful.	89%	1	0	1	6	12	
The 'new facts' in the information box helped me learn new syntax and concepts.	95%	0	0	1	6	13	
The 'gotchas' in the information box were helpful in learning about potential pitfalls.	93%	0	2	0	6	12	
The code output box helped me understand new syntax in R.	79%	3	0	1	8	8	
I found opunit unnecessarily complex.	0%	1	3	0	0	0	

[1] Likert responses: Strongly Disagree (SD), Disagree (D), Neutral (N), Agree (A), Strongly Agree (SA).
[2] Net stacked distribution removes the Neutral option and shows the skew between positive (more useful) and negative (less useful) responses.
Strongly Disagree, Disagree, Agree; Strongly Agree.

of the comma blank?" Another participant wondered about the meaning of a negative sign in front of R's `order` function by expressing they *"don't get why the minus sign is there."* [P8].

Tool highlighted facts participants may have misunderstood or missed: The highlighting of the syntax elements and stepping through the code incrementally helped participants focus on the important parts of the code snippets. For additional feedback, one participant said *"I was rarely confused by the descriptions, and the colorized highlighting helped me keep track of my thoughts and reference what exactly it was I was reading about with a specific example"* [P17]. P13 had a similar feedback remarking that the *"highlighting was good since most people just try to summarize the whole code at once."* However, a few participants found the stepper to progress the lesson too slowly. P17 read the entire line of code on the 'Selecting rows and sorting' lesson and said that they *"didn't understand drop=FALSE, hasn't been mentioned"* before Transfer Tutor had the opportunity to highlight it.

Reluctance of accepting facts without execution or examples: Participants were reluctant to accept certain facts without confirming for themselves through code execution, or without seeing additional examples. One participant was *"not too sold on the explanation"* [P2] for why parentheses aren't required around conditions when subsetting data frames. Another participant expressed doubt and confusion when reading about an alternate `[` syntax that doesn't require specifying both rows and columns: *"Ok but then it says you can use an alternate syntax without using the comma"* [P20]. Regarding the code output, one participant suggested that *"it would've been more useful if I could change [the code] live and observe the output"* [P18]. There were a few participants who wanted more examples. For example, P17 was unclear on how to use the `[[` syntax in R and suggested that *"maybe if there was a specific example here for the [[that would help"*.

Information overload: Although several participants reported that Transfer Tutor is *"interactive and easy to use"* [P13], there were a few who thought that there was *"information overload in the textual explanations"* [P1]. Some syntax elements had lengthy explanations and one participant felt that *"sometimes too many new things were introduced at once"* [P18] and P5 expressed that *"complex language is used"* to describe a syntax or concept in R. Participants also expressed that they wanted *"more visual examples"* [P5].

C. RQ3: How satisfied are programmers with learning R when using Transfer Tutor?

Table II shows the distribution of responses for each statement from the user satisfaction survey, with each statement targeting a feature of Transfer Tutor. Overall, participants indicated that features of Transfer Tutor were useful in learning R. However, a few participants strongly disagreed about the usefulness of explanations relating back to R, and the output boxes. The free-form responses from participants offers additional insight into the Likert responses which will be discussed next.

The *highlighting feature* had no negative ratings and all participants indicated that it was useful to them in some way. One participant thought that *"the highlighting drew [their] attention"* [P2] while another commented that *"it showed the differences visually and addressed almost all my queries"* [P1].

The *stepper* received some neutral (3) ratings and one participant disagreed on its usefulness. Nevertheless, most participants did find the stepper useful and expressed that they *"like how it focuses on things part by part"* [P20].

Participants generally found the *explanations* relating R to Python was useful in learning R. One of the participants *"liked the attempt to introduce R syntax based on Python syntax"* [P18] and P14 thought that *"comparing it with Python makes it even more easy to understand R language"*. All participants thought this feature was useful except for one. This participant did not provide any feedback for why.

The 'new facts' explanations also had no negative ratings and was useful to all participants. Although participants didn't speak explicitly about the feature, P8 expressed that there was *"detailed explanation for each element"* and P16 said that *"Every aspect of the syntax changes has been explained very*

well". Most participants also found 'gotchas' to be useful. P7 for example said that *"Gotchas! were interesting to learn and to avoid errors while coding."*

For the *explanation box*, some participants suggested that this affordance would need to *"reduce the need for scrolling and (sadly) reading"* [P2]. Still other participants wanted deeper explanations for some concepts, perhaps with *"links to more detailed explanations"* [P12]. For the *output boxes*, participants who disagreed with its usefulness suggested that the output boxes would be more useful if the output code be dynamically adjusted by changing the code [P6, P9, P12], and P17 suggested that the output boxes were *"a little difficult to read"* because of the small font.

VI. Limitations

A. Construct Validity

We used pre-test and post-test questions as a proxy to assess the participants' understanding of R concepts as covered by Transfer Tutor. Because of time constraints in the study, we could only ask a limited number of questions. Consequently, these questions are only approximations of participants' understanding. For instance, Question 7 illustrates several reasons why questions may be problematic for programmers. First, the question may be confusingly-worded, because of the use of *except* in the question statement. Second, the response may be correct, but incomplete—due to our scoring strategy, responses must be completely correct to receive credit. Third, questions are only approximations of the participants' understanding. A comparative study is necessary to properly measure learning from using Transfer Tutor to other traditional methods of learning languages by measuring performance on programming tasks.

B. Internal Validity

Participants in the study overwhelmingly found the features of Transfer Tutor to be positive (Section V-C). It's possible, however, that this positivity is artificially high due to social desirability bias [23]—a phenomenon in which participants tend to respond more positively in the presence of the experimenters than they would otherwise. Given the novelty of Transfer Tutor, it is likely that they assumed that the investigator was also the developer of the tool. Thus, we should be conservative about how we interpret user satisfaction with Transfer Tutor and its features.

A second threat to internal validity is that we expected Transfer Tutor to be used by experts in Python, and novices in R. Although all of our participants have limited knowledge with R, very few participants were also experts with Python or the Pandas library (Section IV). On one hand, this could suggest that learning transfer would be even more effective with expert Python/Pandas participants. On the other hand, this could also suggest that there is a confounding factor that explains the increase in learning that is not directly due to the tool. For instance, it may be that explanations in general are useful to participants, whether or not they are phrased in terms of transfer [24], [25].

C. External Validity

We recruited graduate students with varying knowledge of Python and R, so the results of the study may not generalize to other populations, such as industry experts. The choice of Python and R, despite some notable differences, are both primarily intended to be used as scripting languages. How effective language transfer can be when language differences are more drastic is still an open question; for example, consider if we had instead used R and Rust—languages with very different memory models and programming idioms.

VII. Design Implications

This section presents the design implications of the results and future applications for learning transfer.

A. Affordances for supporting learning transfer

Stepping through each line incrementally with corresponding highlighting updates allows programmers to focus on the relevant syntax elements for source code. This helps novice programmers pinpoint misconceptions that could be easily overlooked otherwise, but prevents more advanced programmers from easily skipping explanations from Transfer Tutor. Despite the usefulness of always-on visualizations in nice environments [26], [27], an alternative implementation approach to always-on may be to interactively allow the programmer to activate explanations on-demand.

We found that live code execution is an an important factor for programmers as they can test new syntax rules or confirm a concept. We envision future iterations of Transfer Tutor that could allow code execution and adapt explanations in the context of the programmers' custom code.

Reducing the amount of text and allowing live code execution were two improvements suggested by the participants. This suggests that Transfer Tutor needs to reduce information overload and balance the volume of explanation against the amount of code to be explained. One solution is to externalize additional explanation to documentation outside of Transfer Tutor, such as web resources. Breaking up lessons into smaller segments could also reduce the amount of reading required.

B. Expert learning can benefit from learning transfer

To prevent negative consequences for experienced learners, we intentionally mitigated the expertise reversal effect [4] by presenting explanations in terms of language transfer—in the context of a language that the programmer is already an expert at. Participants in our study tried to guess positive transfers on their own, which could lead to negative transfers from their previous languages. This cognitive strategy is better supported by a tool like Transfer Tutor as it guides programmers on the correct positive transfers and warns them about potential negative transfers. We think that tools such as ours serves as a type of intervention design: like training wheels, programmers new to the language can use our tool to familiarize themselves with the language. As they become experts, they would reduce and eventually eliminate use of Transfer Tutor.

C. Learning transfer within programming languages

Our study explored learning transfer between programming languages, but learning transfer issues can be found within programming languages as well, due to different programming idioms within the same language. For example, in the R community, a collection of packages called `tidyverse` encourage an opinionated programming style that focuses on consistency and readability, through the use of a *fluent* design pattern. In contrast to 'base' R—which is usually structured as a sequence of data transformation instructions on data frames—the fluent pattern uses 'verbs' that pipe together to modify data frames.

D. Applications of learning transfer beyond tutorials

Learning transfer could be applied in other contexts, such as within code review tools, and within integrated development environments such as Eclipse and Visual Studio. For example, consider a scenario in which a software engineer needs to translate code from one programming language to another: this activity is an instance in which learning transfer is required. Tools could assist programmers by providing explanations in terms of their expert language through existing affordances in development environments. Learning transfer tools can be beneficial even when the language conversion is automatic. For example, SMOP (Small Matlab and Octave to Python compiler) is one example of a transpiler—the system takes in Matlab/Octave code and outputs Python code.[3] The generated code could embed explanations of the translation that took place so that programmers can better understand why the translation occurred the way that it did.

Another potential avenue for supporting learning transfer with tools can be found in the domain of documentation generation for programming languages. Since static documentation can't support all types of readers, authors make deliberate design choices to focus their documentation for certain audiences. For example, the canonical Rust book[4] makes the assumption that programmers new to Rust have experience with some other language—though it tries not to assume any particular one. Automatically generating documentation for programmers tailored for prior expertise in a different language might be an interesting application for language transfer.

VIII. Related Work

There are many studies on transfer between tools [28], [29], [30], [31] but fewer studies examining transfer in programming. Transfer of declarative knowledge between programming languages has been studied by Harvey and Anderson [20], which showed strong effects of transfer between Lisp and Prolog. Scholtz and Wiedenbeck [11] conducted a think-aloud protocol where programmers who were experienced in Pascal or C tried implementing code in Icon. They demonstrated that programmers could suffer from negative transfer of programming languages. Wu and Anderson conducted a similar study on problem-solving transfer, where programmers who had experience in Lisp, Pascal and Prolog wrote solutions to programming problems [12]. The authors found positive transfer between the languages which could improve programmer productivity. Bower [32] used a new teaching approach called Continual And Explicit Comparison (CAEC) to teach Java to students who have knowledge of C++. They found that students benefited from the continual comparison of C++ concepts to Java. However, none of these studies investigated tool support.

Fix and Wiedenbeck [14] developed and evaluated a tool called ADAPT that teaches Ada to programmers who know Pascal and C. Their tool helps programmers avoid high level plans with negative transfer from Pascal and C, but is targeted at the planning level. Our tool teaches programmers about negative transfers from Python, emphasizing both syntax and semantic issues by highlighting differences between the syntax elements in the code snippets of the two languages. Transfer Tutor also covers pitfalls in R that doesn't relate to Python.

We leverage existing techniques used in two interactive learning tools for programming, namely Python Tutor [33] and Tutorons [34]. Python Tutor is an interactive tool for computer science education which allows the visualization and execution of Python code. We borrowed the idea of Python Tutor's ability to step through the code and pointing to the current line the program is executing to help the programmer stay focused. Head et al. designed a new technique of generating explanations or *Tutorons* that helps programmers learn about code snippets on the web browser by providing pop-ups with brief explanations of user highlighted code [34]. Although our tool does not automatically generate explanations for highlighted code, it uses the idea of providing details about syntax elements as the programmer steps through the syntax elements which are already highlighted for them.

IX. Conclusion

In this paper, we evaluated the effectiveness of using learning transfer through a training tool for expert Python developers who are new to R. We found that participants were able to learn basic concepts in R and they found Transfer Tutor to be useful in learning R across a number of affordances. Observations made in the think-aloud study revealed that Transfer Tutor highlighted facts that were easy to miss or misunderstand and participants were reluctant to accept certain facts without code execution. The results of this study suggest opportunities for incorporating learning transfer feedback in programming environments.

Acknowledgements

This material is based in part upon work supported by the National Science Foundation under Grant Nos. 1559593 and 1755762.

References

[1] S. E. Sim and R. C. Holt, "The ramp-up problem in software projects: A case study of how software immigrants naturalize," in *International Conference on Software Engineering (ICSE)*, 1998, pp. 361–270.

[3] https://github.com/victorlei/smop

[4] https://doc.rust-lang.org/book/second-edition/

[2] D. Loksa, A. J. Ko, W. Jernigan, A. Oleson, C. J. Mendez, and M. M. Burnett, "Programming, problem solving, and self-awareness: Effects of explicit guidance," in *Human Factors in Computing Systems (CHI)*, 2016, pp. 1449–1461.

[3] L. M. Berlin, "Beyond program understanding: A look at programming expertise in industry," *Empirical Studies of Programmers (ESP)*, vol. 93, no. 744, pp. 6–25, 1993.

[4] S. Kalyuga, P. Ayres, P. Chandler, and J. Sweller, "The expertise reversal effect," *Educational Psychologist*, vol. 38, no. 1, pp. 23–31, 2003.

[5] P. Burns. (2012) The R Inferno. [Online]. Available: http://www.burns-stat.com/documents/books/the-r-inferno/

[6] T. Smith and K. Ushey, "aRrgh: a newcomer's (angry) guide to R," http://arrgh.tim-smith.us.

[7] D. N. Perkins, G. Salomon, and P. Press, "Transfer of learning," in *International Encyclopedia of Education*. Pergamon Press, 1992.

[8] P. Pirolli and M. Recker, "Learning strategies and transfer in the domain of programming," *Cognition and Instruction*, vol. 12, no. 3, pp. 235–275, 1994.

[9] C. M. Kessler, "Transfer of programming skills in novice LISP learners," Ph.D. dissertation, Carnegie Mellon University, 1988.

[10] N. Pennington, R. Nicolich, and J. Rahm, "Transfer of training between cognitive subskills: Is knowledge use specific?" *Cognitive Psychology*, vol. 28, no. 2, pp. 175–224, 1995.

[11] J. Scholtz and S. Wiedenbeck, "Learning second and subsequent programming languages: A problem of transfer," *International Journal of Human–Computer Interaction*, vol. 2, no. 1, pp. 51–72, 1990.

[12] Q. Wu and J. R. Anderson, "Problem-solving transfer among programming languages," Carnegie Mellon University, Tech. Rep., 1990.

[13] L. C. Kaczmarczyk, E. R. Petrick, J. P. East, and G. L. Herman, "Identifying student misconceptions of programming," in *Computer Science Education (SIGCSE)*, 2010, pp. 107–111.

[14] V. Fix and S. Wiedenbeck, "An intelligent tool to aid students in learning second and subsequent programming languages," *Computers & Education*, vol. 27, no. 2, pp. 71 – 83, 1996.

[15] "Stack Overflow," https://stackoverflow.com.

[16] R. K. Sawyer, *The Cambridge Handbook of the Learning Sciences*. Cambridge University Press, 2005.

[17] B. Victor. (2012) Learnable programming. [Online]. Available: http://worrydream.com/LearnableProgramming/

[18] J. G. Greeno, J. L. Moore, and D. R. Smith, "Transfer of situated learning." in *Transfer on trial: Intelligence, cognition, and instruction.* Westport, CT, US: Ablex Publishing, 1993, pp. 99–167.

[19] D. H. Schunk, *Learning Theories: An Educational Perspective*, 6th ed. Pearson, 2012.

[20] L. Harvey and J. Anderson, "Transfer of declarative knowledge in complex information-processing domains," *Human-Computer Interaction*, vol. 11, no. 1, pp. 69–96, 1996.

[21] A. Ohri, *Python for R Users: A Data Science Approach*. John Wiley & Sons, 2017.

[22] D. Spencer, *Card Sorting: Designing Usable Categories*. Rosenfeld, 2009.

[23] N. Dell, V. Vaidyanathan, I. Medhi, E. Cutrell, and W. Thies, "'Yours is better!': Participant response bias in HCI," in *Human Factors in Computing Systems (CHI)*, 2012, pp. 1321–1330.

[24] T. Kulesza, S. Stumpf, M. Burnett, S. Yang, I. Kwan, and W.-K. Wong, "Too much, too little, or just right? Ways explanations impact end users' mental models," in *Visual Languages and Human-Centric Computing (VL/HCC)*, 2013, pp. 3–10.

[25] A. Bunt, M. Lount, and C. Lauzon, "Are explanations always important?" in *Intelligent User Interfaces (IUI)*, 2012, pp. 169–178.

[26] H. Kang and P. J. Guo, "Omnicode: A novice-oriented live programming environment with always-on run-time value visualizations," in *User Interface Software and Technology (UIST)*, 2017, pp. 737–745.

[27] J. Hoffswell, A. Satyanarayan, and J. Heer, "Augmenting code with in situ visualizations to aid program understanding," in *Human Factors in Computing Systems (CHI)*, 2018, pp. 532:1–532:12.

[28] P. G. Polson, "A quantitative theory of human-computer interaction," in *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, 1987, pp. 184–235.

[29] P. G. Polson, S. Bovair, and D. Kieras, "Transfer between text editors," in *Human Factors in Computing Systems and Graphics Interface (CHI/GI)*, vol. 17, no. SI, 1986, pp. 27–32.

[30] P. G. Polson, E. Muncher, and G. Engelbeck, "A test of a common elements theory of transfer," in *Human Factors in Computing Systems (CHI)*, vol. 17, no. 4, 1986, pp. 78–83.

[31] M. K. Singley and J. R. Anderson, "A keystroke analysis of learning and transfer in text editing," *Human-Computer Interaction*, vol. 3, no. 3, pp. 223–274, 1987.

[32] M. Bower and A. McIver, "Continual and explicit comparison to promote proactive facilitation during second computer language learning," in *Innovation and Technology in Computer Science Education (ITiCSE)*, 2011, pp. 218–222.

[33] P. J. Guo, "Online Python Tutor: Embeddable web-based program visualization for cs education," in *Computer Science Education (SIGCSE)*, 2013, pp. 579–584.

[34] A. Head, C. Appachu, M. A. Hearst, and B. Hartmann, "Tutorons: Generating context-relevant, on-demand explanations and demonstrations of online code," in *Visual Languages and Human-Centric Computing (VL/HCC)*, 2015, pp. 3–12.